05 | 一个好的自动化测试长什么样？

2021-08-13 郑晔

《程序员的测试课》 课程介绍 >

讲述：郑晔

时长 11:44 大小 10.75M

你好，我是郑晔！

在上一讲里我们讲了测试的一个关键点是自动化测试，而自动化刚好是程序员的强项。自从有了自动化测试框架，自动化测试就从业余走向了专业，但这并不是说，有了测试框架你就能把测试写好了，我们来看几个典型的问题：

测试不够稳定，一次运行通过，下次就不能通过了；

要测的东西很简单，但是为了测这个东西，光是周边配套的准备就要写很多的代码

一个测试必须在另一个测试之后运行；

……

这是让很多团队在测试中挣扎的原因，也是很多人放弃测试的理由。之所以测试会出现这样那样的问题，一个重要的原因是这些测试不够好。这一讲，我们就来讲讲好的测试应该长什么样。

测试的样子

关于自动化测试，其实有一个关键的问题我们一直还没有讨论。我们用测试来保证代码的正确性，然而，测试的正确性如何保证呢？

这是一个会问懵很多人的问题：测试保证代码的正确性，那测试代码的正确性也用测试保证？但你见过有人给测试写测试吗？没有。因为这是一个循环的问题，你给测试写了测试，那新的测试怎么保证正确性呢？难不成要递归地写下去？是不是有种大脑要堆栈溢出的感觉了。

既然给测试写测试不是一个行得通的做法，那唯一可行的方案就是，**把测试写简单，简单到一目了然，不需要证明它的正确性**。由此，我们可以知道，一个复杂的测试肯定不是一个好的测试。

简单的测试应该长什么样呢？我们一起来看一个例子，这就是我们在实战环节中给出的第一个测试。

复制代码

```
@Test
public void should_add_todo_item() {
  // 准备
  TodoItemRepository repository = mock(TodoItemRepository.class);
  when(repository.save(any())).then(returnsFirstArg());
  TodoItemService service = new TodoItemService(repository);

  // 执行
  TodoItem item = service.addTodoItem(new TodoParameter("foo"));

  // 断言
  assertThat(item.getContent()).isEqualTo("foo");

  // 清理(可选)

}
```

我把这个测试分成了四段，分别是准备、执行、断言和清理，这也是一般测试都会具备的四个阶段，我们分别来看一下。

准备。这个阶段是为了测试所做的一些准备，比如启动外部依赖的服务，存储一些预置的数据。在我们这个例子里面就是设置所需组件的行为，然后将这些组件组装了起来。

执行。这个阶段是整个测试中最核心的部分，触发被测目标的行为。通常来说，它就是一个测试点，在大多数情况下，执行应该就是一个函数调用。如果是测试外部系统，就是发出一个请求。在我们这段代码里，它就是调用了一个函数。

断言。断言是我们的预期，它负责验证执行的结果是否正确。比如，被测系统是否返回了正确的应答。在这个例子，我们验证的是 Todo 项的内容是否是我们添加进去的内容。

清理。清理是一个可能会有的部分。如果在测试中使到了外部资源，在这个部分要及时地释放掉，保证测试环境被还原到一个最初的状态，就像什么都没发生过一样。比如，我们在测试过程中向数据库插入了数据，执行之后，要删除测试过程中插入的数据。一些测试框架对一些通用的情况已经提供支持，比如之前我们用到的临时文件。

如果准备和清理的部分是在几个测试用例间通用的，它们就有可能被放到 setUp 和 tearDown 里去完成，这一点我们在上一讲已经讲过了。

这四个阶段中，必须存在的是执行和断言。想想也是，**不执行，目标都没有，还测什么？不断言，预期都没有，跑了也是白跑。**如果不涉及到一些资源释放，清理部分很可能就没有了。而对一些简单的测试来说，也不需要做特别的准备。

从结构上来看，测试用例应该就是这么简单。你去看一下我们在实战中的代码，大部分测试都是可以这样划分的。

理解了测试的结构，有一些测试存在的问题你一眼就能看出来了。比如对于没有断言的测试来说，看上去测试从来不会出错，但这样的测试几乎是没有价值的。

再比如，一个测试里有多个执行目标，可能是需要在一个测试里要测多个不同的函数。这就是一个坏味道了。为什么说这是一个坏味道呢？因为测试的根基是简单，一旦复杂了，我们就很难保证测试本身的正确性。如果你有多个目标怎么办？分成多个测试就好了。

如果测试本身简单到令人发指的程度，出于节省代码篇幅的角度，你可以考虑在一个测试里面写。比如测试字符串为空的函数，我要分别传入空对象和空字符串，每种情况执行和断言一行代码就写完了，那我可能就在一个测试里面写了。

一段旅程（A-TRIP）

有了对测试结构的基本认知，我们再进一步，看看如何衡量一个测试有没有做好？有人把好测试的特点总结成一个说法：**A-TRIP**。这其实是五个单词的缩写，分别是：

Automatic，自动化；

Thorough，全面的；

Repeatable，可重复的；

Independent，独立的；

Professional，专业的。

这是什么意思呢？我们分别来解释一下。

Automatic，自动化。经过上一讲的讲解，这一点你应该已经很容易理解了。自动化测试相比传统测试，核心增强就在自动化上。这也是为什么测试一定要有断言，因为只有在有断言的情况下，机器才能够帮我们判断测试是否成功。

Thorough，全面的。这一点其实是测试的要求，应该尽可能用测试覆盖各种场景。不管什么样的自动化测试，它的本质还是测试，前面我们讲了向测试人员学习，关键点就在于这有助于我们写出更全面的测试。理解全面还有一个角度，就是测试覆盖率。我们在实战环节中已经见识了如何通过测试覆盖率工具，帮我们去发现代码中测试中没有覆盖到地方。

Repeatable，可重复的。它要求测试能够反复运行，并且结果都应该是一样的。这是保证测试简单可靠的前提。如果一个测试不是可重复的，我们就没法相信它的运行结果，测试的价值也就荡然无存了。一旦测试报错，我们没法确定是我们程序出错了，还是其它什么地方出错了。

在内存中执行的测试一般都是可重复的。影响一个测试可重复性的主要因素是外部资源，常见的外部资源包括文件、数据库、中间件、第三方服务等等。如果在测试中遇到这些外部资源，我们就要想办法让这些资源在测试结束后，恢复原来的样子。你在实战中已经见识过如何处理文件，在后面的应用篇，我们还会讲到如何处理数据库。简单说就是在测试执行之后，能够把数据回滚掉。

如果你遇到中间件，最好有一个独立可控的中间件。而遇到第三方服务，则可以采用模拟服务，我的开源项目 Moco 主要就是为了解决这种外部依赖而生的。

理解可重复性还有一个角度，那就是一批测试也要可重复。这就需要测试之间彼此没有依赖，这也是我们接下来要讨论的测试的另外一个特点。

Independent，独立的。测试和测试之间不应该有任何依赖。什么叫有依赖？就是一个测试要依赖于另外一个测试运行的结果。比如两个测试都要依赖于数据库，第一个测试运行时往数据库里写了一些数据，而第二个测试在执行时要用到这些数据。也就是说，第二个测试必须在第一个测试执行之后再执行，这就叫做有依赖。

我知道，有很多人有很多的理由让测试之间有依赖。比如说为了提高执行效率，但这种做法属于特定的优化。对于其他绝大多数情况而言，一旦你开始这么做了，测试就走上了歧途。比如，一些框架支持多个测试并行运行，一旦测试有依赖，测试就无法并行执行，因为这两个测试之间是有顺序的。再比如，一旦有人破坏了测试的独立性，紧接着就会有更多的人破坏独立性，这就像代码的坏味道一样，很容易传播。

可重复性和独立性关联非常紧密。因为我们通常认为，可重复是测试按照随机的顺序执行，其结果也是一样的，这就要依赖于测试是独立的。而一旦测试不独立，有了依赖，从单个测试上来看，它也违反了可重复性。

Professional，专业的。这一点是很多人观念中缺失的，测试代码也是代码，也要按照代码的标准去维护。这就意味着你的测试代码也要写得清晰，比如良好的命名、把函数写小、要重构甚至要抽象出测试的基础库、测试的模式。在 Web 测试中常见的 PageObject 模式，就是这种理念的延伸。

有一点我准备多说几句，就是测试的命名。很多人写代码时，知道要取一个有意义的命名，但在测试上常常忽略这一点，我们经常可以看到 test1、test2 这样的测试命名。那测

试应该怎么命名呢？

我不知道你是否注意到了，我在实战中写的测试，其命名与传统的 Java 函数有着很大的区别。首先，我用了下划线区隔单词，而没有采用驼峰命名；其次，名字都很长；再有，所有的测试都是以 should 开头。

我为什么要这么写呢？其实，我是希望在测试名中把测试用例的场景给描述出来。换言之，这个测试名不是一个简单的名字，而是一句话，这样测试的名字就会很长。而一旦名字太长，用驼峰阅读起来就不那么舒服了，所以，我采用了下划线区隔。

我对测试的命名主要有两种：

should_ 测试场景；

should_ 测试效果 _while_ 测试条件。

第一种命名表示应该做成什么样子，比如，should_add_todo_item，一般来说，对于一个正常情况的测试用例，我会这么命名。第二种情况则表示在什么条件下，应该出现什么效果，比如，should_throw_exception_while_parameter_is_empty，可以用来描述各种异常的情况。你会看到这两种命名方法其实都是写了一句话，而之所以会用 should 开头，它表示这个测试“应该”是什么样的。

有一些测试框架在测试描述上做得更加进一步，看上去就更像一句话了，下面是一个例子。

复制代码

```
1 it.should("throw exception while parameter is empty", () -> {
2   ...
3 });
```

经过这一讲的介绍，相信你对一个好的测试应该长成什么样已经有了一个初步的认识，但知道了好测试长什么样，只能帮助你发现测试中存在的问题。下一讲，我们接着来讨论一个影响写好测试的关键因素：软件设计。

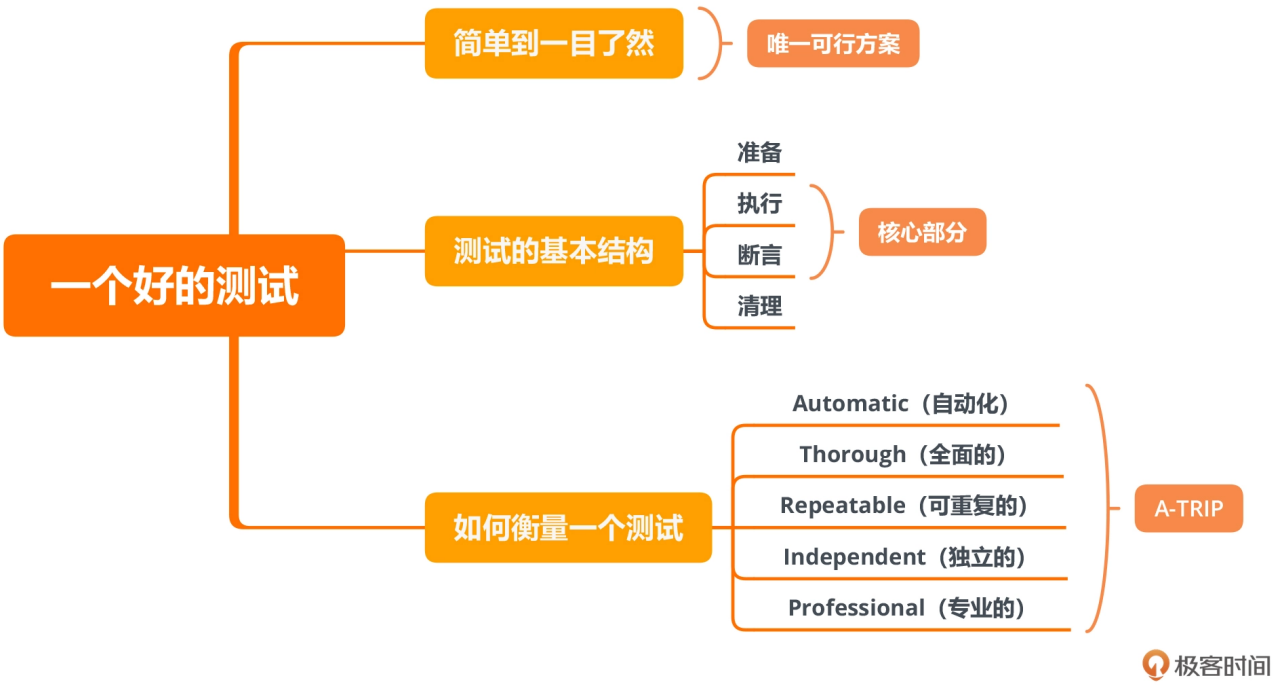
总结时刻

这一讲，我们讨论了一个好的测试应该是什么样子的。一个好的测试首先应该是简单的，否则，我们无法保证测试的正确性。

我们还谈到了测试的基本结构：准备、执行、断言和清理。其中，核心的部分是执行和断言。一个测试既不能执行太多的东西，也不能没有断言。

怎么衡量测试是否做好了呢？有一个标准：A-TRIP，这是五个单词的缩写，分别是 Automatic（自动化）、Thorough（全面的）、Repeatable（可重复的）、Independent（独立的）和 Professional（专业的）。

如果今天的内容你只能记住一件事，那请记住：**编写简单的测试**。



思考题

用今天讲到好测试的原则去对比一下你的测试，你会发现哪些问题呢？欢迎在留言区分享你的发现。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

赞 1 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 04 | 自动化测试：为什么程序员做测试其实是有优势的？

下一篇 06 | 测试不好做，为什么会和设计有关系？

精选留言 (7)

写留言

亦无
2021-08-18

郑老师对测试理解的很到位，提到的几个理念都非常的准确，而且还做了很精炼的总结，真给力。

1、每一条测试的测试目的必须非常准确，比如我之前推荐的测试用例写法就是「验证XXXX」，测试目的一目了然；...

展开

Alexdown
2021-08-17

举例：一个对象有如下两个方法

write(p []byte)(n int, err error) 将byte数组p写入对象，返回写入的byte个数n及在此过程中可能出现的错误err
read(q []byte)(n int, err error)将对象内容写入byte数组q，返回写入的byte个数n及在...

展开

大茄子实验室
2021-08-16

背景：从事数仓开发，没有对应的测试人员，也没有使用过测试框架。一般都是自己写测试脚本进行比对
测试后，清理会延迟做，因为一天的数据对，不代表每天的数据对，备份数据会保留一段时间。
问题：...

展开

作者回复: 多谢分享

¦并向你招手¦样子

2021-08-14

Junit5 的@DisplayName 注解，可以文档化测试的内容，用它说明测试的内容和条件可以让测试更友好一些

作者回复: 多谢分享

邓志国

2021-08-14

我喜欢有时候用中文命名测试方法，感觉更容易阅读理解

作者回复: 这取决于团队对于英文水平的要求

davix

2021-08-13

請問那種協議交互的場景：來回幾個消息的一系列過程，用什麼結構好？

作者回复: 把业务和技术实现隔离开，单独测业务，再单独测通信。

1 1

William

2021-08-13

展开